Datorlaboration 8

Måns Magnusson och Josef Wilzén

1 april 2018

```
## Loading required package: methods
## pxweb: R tools for PX-WEB API.
## Copyright (C) 2014 Mans Magnusson, Leo Lahti, Love Hansson
## https://github.com/ropengov/pxweb
## New PXWEB API(s):
##  - prognos.konj.se
##  - data.ssb.no
##  - pxweb.asub.ax
##  - makstat.stat.gov.mk
##  - data.csb.gov.lv
##
## Use update_pxweb_apis() to update the api catalogue.
```

Instruktioner

- Denna laboration ska göras i grupper om **två och två**. Det är viktigt för gruppindelningen att inte ändra grupper.
- En av ska vara **navigatör** och den andra **programmerar**. Navigatörens ansvar är att ha ett helhetsperspektiv över koden. Byt position var 30:e minut. Båda ska vara engagerade i koden.
- Det är tillåtet att diskutera med andra grupper, men att plagiera eller skriva kod åt varandra är **inte tillåtet**.
- Använd inte `å`, `ä` eller `ö` i variabel- eller funktionsnamn.
- Utgå från laborationsfilen, som går att ladda ned **här**, när du gör inlämningsuppgifterna.
- Spara denna som `labb[no]_grupp[no].R` , t.ex. `labb5_grupp01.R` om det är laboration 5 och ni tillhör grupp 1. Ta inte med hakparenteser eller stora bokstäver i filnamnet.
  Obs! Denna fil ska laddas upp på LISAM och ska **inte** innehålla något annat än de aktuella funktionerna, namn-, ID- och grupp-variabler och ev. kommentarer. Alltså **inga** andra variabler, funktionsanrop för att testa inlämningsuppgifterna eller anrop till markmyassignment-funktioner.
- Om ni ska lämna i kompletteringar på del 2, döp då dessa till `labb5_grupp01._komp1.R` om det är första kompletteringstillfället. Se kurshemsidan för mer information om kompletteringar.
- Laborationen består av två delar:
  - Datorlaborationen
  - Inlämningsuppgifter
- I laborationen finns det extrauppgifter markerade med *. Dessa kan hoppas över.
- Deadline för laboration framgår på **LISAM**
- **Tips!** Använd "fusklapparna" som finns **här**. Dessa kommer ni också få ha med på tentan.

Innehåll

Del I

Datorlaboration

Kapitel 1

Texthantering och regular expression i R med `stringr`

Gör följande delar i *Handling and Processing Strings in R* av Gaston Sanchez.

Kap 2 Hela

Kap 3 3.1, 3.3

Kap 4 4.2.1 - 4.2.3

Kap 5 5 - 5.2.2, 5.2.6, 5.3.1-5.3.2

Kap 6 6-6.1.3, 6.2.2, 6.4 - 6.4.1, 6.4.3, 6.4.5, 6.4.7, 6.4.9 - 6.4.10

Boken finns fritt tillgänglig **[här]**.

Kapitel 2

Modern datahantering

I följande kapitel gås de verktyg som idag är state-of-the-art för att snabbt och effektivt bearbeta stora datamängder i R.

2.1 Piping med %>%

När vi arbetar med databearbetning av stora datamaterial kan innebär det ofta ett stort antal funktionsanrop. För att göra en databearbetningsprocess överskådlig och snabb finns så kallade pipes, eller ”rör” i R för att skicka datamaterial i ett flöde av olika modifikationer. Pipingoperatorn innbär att

```
z <- a %>% fun1(b) %>% fun2(c) %>% fun3()
```

är exakt samma sak som

```
x <- fun(a, b)
y <- fun(x, c)
z <- fun3(y)
```

Detta flöde kan ofts göra det tydligare hur data bearbetas i ett databearbetningssteg.

2.2 `tidyr`

Datamaterial kan många gånger komma i ett otal olika tabellstrukturer. Alla statistiska metoder, databearbetningsverktyg som `dplyr` och visualiseringspaket som `ggplot2` kräver att datamaterialet är i ett så kallat `tidy` format. För att konvertera olika tabeller och datamaterial till ett `tidy` format används paketet `tidyr` och funktionen `gather()`.

Gå igenom och reproducera koden i följande **introduktionstext**.

2.3 `dplyr`

R-paketet dplyr har under kort tid att bli det huvudsakliga verktyget för att arbeta med större datamängder i R. Det finns framförallt tre anledningar till dess popularitet.

1. Paketet har bara ett fåtal funktioner för att arbeta med data vilket gör det snabbt att lära sig.
2. `dplyr` är skrivet i kraftigt optimerad C++ kod vilket gör hanteringen av stora datamängder snabbare än något annat statistik- eller analysverktyg.
3. `dplyr` kan kopplas mot databaser för att direkt bearbeta större datamängder. dplyrs verb används också i `sparlyr`, vilket är ett paket för att hantera data som inte får plats på enskilda datorer. Att lära sig `dplyr` är således en approach som möjliggör att hantera i princip hur stora datamaterial som helst.

Gå igenom och reproducera koden i följande **introduktionstext**. Hoppa över avsnittet **Other data sources**.

Del II

Inlämningsuppgifter

Tips!

Inlämningsuppgifterna innebär att konstruera funktioner. Ofta är det bra att bryta ned programmeringsuppgifter i färre små steg och testa att det fungerar i varje steg.

1. Lös uppgiften med vanlig kod direkt i R-Studio (precis som i datorlaborationen ovan) utan att skapa en funktion.
2. Testa att du får samma resultat som testexemplen.
3. Implementera koden du skrivit i 1. ovan som en funktion.
4. Testa att du får samma resultat som i testexemplen, nu med funktionen.

Automatisk återkoppling med `markmyassignment`

Som ett komplement för att snabbt kunna få återkoppling på de olika arbetsuppgifterna finns paketet `markmyassignment`. Med detta är det möjligt att direkt få återkoppling på uppgifterna i laborationen, oavsett dator. Dock krävs internetanslutning.

Information om hur du installerar och använder `markmyassignment` för att få direkt återkoppling på dina laborationer finns att tillgå **här**.

Samma information finns också i R och går att läsa genom att först installera `markmyassignment`.

```
install.packages("markmyassignment")
```

Om du ska installera ett paket i PC-pularna så behöver du ange följande:

```
install.packages("markmyassignment",lib="sökväg till en mapp i din hemkatalog")
```

Tänk på att i sökvägar till mappar/filer i R i Windowssystem så används ‘‘\\’’, tex ‘‘C:\\Users\\Josef’’.

Därefter går det att läsa information om hur du använder `markmyassignment` med följande kommando i R:

```
vignette("markmyassignment")
```

Det går även att komma åt vignetten **här**. Till sist går det att komma åt hjälpfilerna och dokumentationen i `markmyassignment` på följande sätt:

```
help(package="markmyassignment")
```

Lycka till!

Kapitel 3

Inlämningsuppgifter

För att använda `markmyassignment` i denna laboration ange:

```
library(markmyassignment)

Loading required package: yaml
Loading required package: testthat
Loading required package: httr
Loading required package: checkmate

lab_path <-
"https://raw.githubusercontent.com/STIMALiU/KursRprgm/master/Labs/Tests/d8.yml"
suppressWarnings(set_assignment(lab_path))

Assignment set:
D8: Statistisk programmering med R: Lab 8
The assignment contain the following (5) tasks:
- messy_swe_pop_handler
- wordcount
- pnr_sex
- pnr_ctrl
- minimum_editing_distance
```

3.1 `messy_swe_pop()`

Nu ska vi arbeta med och hantera lite större datamaterial. Vid laboratoration 5 använde ni paketet `pxweb` för att hämta ned befolkningsstatistik från SCB. Vi ska nu hämta ned all befolkningsstatistik, på den mest detaljerade nivån från SCB. Kör följande kod och ladda ned data. Det kommer ta ett antal minuter att ladda ned.

```
swe_pop_data <-
  get_pxweb_data(
    url = "http://api.scb.se/OV0104/v1/doris/sv/ssd/BE/BE0101/BE0101A/BefolkningNy",
    dims = list(
      Region = c('*'),
      Civilstand = c('*'),
      Alder = c('*'),
      Kon = c('*'),
      ContentsCode = c('BE0101N1'),
      Tid = c('*')),
  clean = FALSE)

# Ta inte med koden ovan i filen ni lämnar in!
```

Notera att `clean` är satt till `FALSE`. Detta är det orginalformat SCB har materialet i, vilket är ett ”messy” format. Fördelen med detta format är att det tar mindre utrymme, vilket gör att det går att ladda ned snabbare.

Skapa nu en funktion du kalla `messy_swe_pop_handler()` som tar den `data.frame` du laddat ovan, gör om den till ett `tidy` format. Sedan ska funktionen aggregera upp variablerna civilstånd och kön och returnera ett dataset men i `dplyr`-formatet `tbl_df` eller `tibble`. Notera att ni kan få ett annat antal rader (beroende på när SCB:s senaste statistik släpps). Men ni bör få samma värden för år 1968 nedan. Ni behöver dock översätta variabelnamnen direkt. Nedan finns en mall för funktionen som ersätter variabelnamnen automatiskt.

```
messy_swe_pop_handler <- function(messy){
  years <- paste0("year", 1968:(1968 + ncol(messy) - 5))
  names(messy) <- c("region", "status", "age", "sex", years)

  # Put your code here

}
```

Tips! Börja med ett mindre dataset (ex. bara år 1968) först. Det tar ett antal sekunder att göra dessa berarbetningar på det stora datamaterialet.

```
messy_swe_pop_handler(messy = swe_pop_data)

Source: local data frame [1,527,552 x 4] Groups: region, year [?]
     region  year     age population
      <chr> <chr>   <chr>  <int>
1  00 Riket  1968    0 år 110889
2  00 Riket  1968    1 år 120179
3  00 Riket  1968   10 år 104883
4  00 Riket  1968 100+ år    105
5  00 Riket  1968   11 år 106466
6  00 Riket  1968   12 år 107469
7  00 Riket  1968   13 år 106731
8  00 Riket  1968   14 år 104580
9  00 Riket  1968   15 år 109081
10 00 Riket  1968   16 år 109148
# ... with 1,527,542 more rows
```

Du ska använda `dplyr` och `tidyr` för att lösa uppgiften. Detta innebär att `for`, `while` och `repeat`-loopar är inte tillåtna. Inte heller `aggregate()`-funktionen eller någon av `apply`-funktionerna.

3.2 `wordcount()`

Nu är uppgiften att skapa en funktion som ska kunna räkna hur många gånger olika ord förekommer i texten. Funktionen ska heta `wordcount()` och ha argumentet `text` som ska vara en `character`-vektor. Funktionen ska ta en text (i form av en text vektor) och returnera en `data.frame` med två variabler `word` (textvariabel) och `freq` (integervariabel).

I variabeln `word` ska respektive ord ingå, men med små bokstäver, och i variabeln `freq` ska frekvensen av orden framgå. Den `data.frame` som returneras ska vara sorterad efter variabeln `word`. Funktionen ska också skriva ut meningen “`The most common word is '[ord]' and it occured [antal] times.`” med `message()`.

Tips! `table()`

Nedan är ett förslag på hur ni kan implementera funktionen.

1. Läs in paktet i `stringr` i den aktuella R-sessionen. OBS: ej installera paktet.
2. Börja med att sätta ihop de olika textelementen till en textsträng, men denna gång använd mellanslag som avskiljare istället för `\n`.

3. Ta bort punkter och kommatecken i textsträngen.
4. Gör om alla ord till endast gemener.
5. Dela upp teckensträngen med `str_split()` för att få ut respektive ord. [**Tips!** Tänk på att du får ut en lista med denna funktion, inte en vektor. `unlist()` kan då vara till hjälp.]
6. Räkna respektive ord och skapa en data.frame med respektive ord i kolumn 1 och antalet förekomster av detta ord i kolumn 2. Döp kolumn 1 till "`word`", och kolumn 2 till "`freq`".
7. Sortera datasetet efter `word`.
8. Använd `str_c()` och `message()` för att baserat på datsetet ovan skriva ut följande mening "`The most common word is '[ord]' and it occured [antal] times.`".
9. Returnera din `data.frame`.

Kolla om testfallen nedan fungerar:

```
# Laddar ned testdata
library(downloader)
transtrommer_remote <-
  "https://raw.githubusercontent.com/STIMALiU/KursRprgm/master/Labs/DataFiles/transtrom.txt"
transtrommer_local <- paste0(getwd(), "/transtrom.txt")
download(url = transtrommer_remote, destfile = transtrommer_local)

# Test
text<-readLines("transtrom.txt")
worddata<-wordcount(text=text)
```

The most common word is 'the' and it occured 8 times.

```
head(worddata)
```

```
        word freq
1          a    6
2        and    4
3 approached    1
4         as    1
5     before    1
6      black    1
```

```
head(worddata[order(worddata[,2], decreasing=TRUE),])
```

```
   word freq
59  the    8
1     a    6
2   and    4
24 have    3
40   of    3
62 they    3
```

```
head(wordcount(text=rep("a",10)))
```

The most common word is 'a' and it occured 10 times.

```
  word freq
1    a   10
```

3.3 Minsta editeringsavstånd

En viktig uppgift i texthantering på datorer är att mäta avstånd mellan olika textsträngar. Vanliga användningsområden är stavningskontroller och optical character recognition (OCR).

Ett vanligt mått på avståndet mellan två strängar är Levenshtein-avståndet, ibland kallat ”minimum editing distance” eller minsta editeringsavstånd. Enkelt uttryckt beskriver detta mått hur många tecken vi måste lägga till, ta bort eller ändra för att gå från den ena strängen till den andra. En introduktion till ”Levenshtein distance” finns på Wikipedia [här].

Exempel:

Levenshtein-avståndet mellan ”hoppa” och ”jobbar” är fyra, då vi behöver ta följande fyra steg:

1. **h**oppa → **j**oppa
2. jo**p**pa → jo**b**pa
3. job**p**a → job**b**a
4. jobba → jobba**r**

Wagner-Fishers algoritm:

Wagner-Fishers algoritm är en metod för att räkna ut Levenshtein-avståndet från två godtyckliga textsträngar. En introduktion till algoritmen samt pseudokod finns på Wikipedia [här]. **Tips!** Läs igenom denna sida och förstå algoritmen innan du börjar. Det finns massa olika implementationer av denna algoritm i andra språk än R. [Här] finns en implementation i språket Go och [här] finns en i Python, de skiljer sig lite från pseudokoden, men kan vara alternativ kod att titta på om man kör fast. [Här]och [här] finns två olika video-material som förklarar algoritmen lite mer utförligt.

Uppgiften är att implementera en funktion kallad `minimum_editing_distance(x, y)` som tar två stycken strängar, `x` och `y` och räknar ut Lewenstein-avståndet mellan dessa två strängar. Funktionen ska avbryta om `x` och `y` inte är textsträngar. Notera att funktionen `adist()` inte är tillåten i denna uppgift. Här är ett testexempel på hur funktionen ska fungera:

```
minimum_editing_distance("jobbar", "hoppa")

[1] 4

minimum_editing_distance("kitten", "sitting")

[1] 3

minimum_editing_distance(" ", "R!")

[1] 2

minimum_editing_distance("Josef Wilzen", "Mans Magnusson")

[1] 12

minimum_editing_distance("programmering", "statistik")

[1] 11

minimum_editing_distance("matematik", "statistik")

[1] 5

minimum_editing_distance("programmering", "matematik")

[1] 11
```

3.4 Svenska personnummer

I Sverige har samtliga medborgare personnummer som de behåller livet ut och som används för identifikation. Personnummret består av tre delar, födelsedatum, födelsenummer och en kontrollsiffra. Som standard anges personnummer på följande sätt `ÅÅÅÅMMDDNNNK` där `ÅÅÅÅ` är födelseåret, `MM` födelsemånaden, `DD` födelsedagen, `NNN` födelsenumret och `K` kontrollsiffran.

Kontrollsiffran beräknas baserat på de övriga siffrorna i personnummret vilket gör att det är möjligt att kontrollera om ett personnummer är korrekt eller inte. Det är också möjligt att utifrån ett personnummer beräkna ålder och kön (samt för vissa även födelseort, men det spelar ingen roll i denna uppgift).

Detaljerna om för hur kön och kontrollsiffran beräknas finns i Skatteverkets broschyr SKV 704 [PDF]. Läs igenom denna broschyr innan du gör uppgiften nedan.

Exempel på personnummer som kan användas för att testa dina funktioner finns dels i broschyren från Skatteverket och dels på Wikipedia (sökord: "Personnummer i Sverige"). Du kan självklart även testa med ditt eget personnummer om du vill.

Vi tar det dock i flera steg, med olika funktioner som utför olika steg. De stegen vi kommer göra är:

1. Skapa en funktion för att kontrollera kontrollsiffran i ett (godtyckligt) personnummer.
2. Skapa en funktion för att ta fram uppgift om kön från ett (godtyckligt) personnummer.

Följande funktioner kommer vara mycket användbara i denna uppgift: `str_c()/paste()`, `str_sub/substr()` och `Sys.Date()`. Kolla upp dessa funktioner innan du sätter igång. Denna funktionalitet för personnummer finns redan för svenska personnummer i paketet `sweidnumbr`, detta paket får inte användas.

Här är ett testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnr_ctrl(pnr)

[1] TRUE

pnr <- "190101010101"
pnr_ctrl(pnr)

[1] FALSE

pnr <- "198112189876"
pnr_ctrl(pnr)

[1] TRUE

pnr <- "190303030303"
pnr_ctrl(pnr)

[1] FALSE
```

3.4.1 Uppgift12: `pnr_ctrl()`

Nästa steg i funktionen är att kontrollera om ett personnummer är korrekt eller inte. För att beräkna en kontrollsiffra används den så kallade Luhn-algoritmen, mer information finns **[här]**. Vi ska skapa en funktion som använder Luhn-algoritmen för att testa om ett personnummer är korrekt eller inte. Utgå från att personnummren kommer på formatet `ÅÅÅÅMMDDNNNK` som en textvektor.

Funktionen ska ta argumentet `pnr` och returnera `TRUE` eller `FALSE` beroende på om personnummret är korrekt eller inte.

Ett förslag på hur funktionen kan implementeras är följande:

1. Dela upp personnummret så respektive siffra blir ett eget element. [**Tips!** `str_split()/strsplit()` och `unlist()`]
2. Konvertera de uppdelade siffrorna till ett numeriskt format.

3. Den vektor av de enskilda siffrorna i personnummret kan nu användas i Luhn - algoritmen. Det enklaste sättet är att multiplicera personnummrets vektor med en beräkningsvektor av 0:or 1:or och 2:or på det sätt som beräkningen specificeras av Luhn-algoritmen.
   Obs! Skatteverkets beräkning görs inte på hela personnummret, de delar som inte ska räknas kan sättas till 0 i beräkningsvektorn.

4. Nästa steg är att summera alla värden i vektorn ovan. Tänk på att tal större än 9 ska räknas som summan av tiotalssiffran och entalssiffran. [**Tips!** `%%` och `%/%`]

5. Summera värdena på vektorn som beräknades i 4 ovan. Plocka ut entalssiffran och dra denna entalssiffra från 10. Du har nu räknat ut kontrollsiffran. Puh!

6. Testa om den uträknade kontrollsiffran är samma som kontrollsiffran i personnummret.

Här är ett testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnr_ctrl(pnr)

[1] TRUE

pnr <- "190101010101"
pnr_ctrl(pnr)

[1] FALSE

pnr <- "198112189876"
pnr_ctrl(pnr)

[1] TRUE

pnr <- "190303030303"
pnr_ctrl(pnr)

[1] FALSE
```

3.4.2 Uppgift 2: `pnr_sex()`

I denna uppgift ska vi från ett personnummer räkna ut det juridiska könet. Som framgår i skattebroschyren ska detta räknas ut genom att undersöka om den näst sista siffran i personnummret är jämt (kvinna) eller udda (man). Detta är vad som definierar en persons juridiska kön.

Skapa nu en funktion du kallar `pnr_sex()` med argumentet `pnr`. Denna funktion ska ta ett personnummer och returnera en persons kön som ett textelement, `M` för man och `K` för kvinna.

Ett förslag på hur funktionen kan implementeras är följande:

1. Plocka ut den näst sista siffran i personnumret.

2. Konvertera denna siffra till numeriskt format och testa om siffran är jämn (returnera `K`) eller udda (returnera `M`)

Här är testexempel på hur funktionen ska fungera:

```
pnr <- "196408233234"
pnr_sex(pnr)

[1] "M"

pnr <- "190202020202"
pnr_sex(pnr)

[1] "K"
```